# Maxima

## Entrées-sorties

`describe("`*cmd*`")` (⇔ `? `*cmd*) `example(`*cmd*`)` `options(`*cmd*`)` `demo("`*chemin*/*fichier*`.dem")`

| | |
|---|---|
| [l]disp[lay](*expr*$_1$,...) | affiche *val*$_1$, ...; *expr*$_1$ =*val*$_1$, ... pour `display`; labels si `ldisp`. |
| `print(`*expr*$_1$,...`)` | affiche les *expr*$_i$ sur 1 ligne; valeur = dernier argument. |
| `batch("`*fichier*`")` | éxécution avec écho du contenu de *fichier*. |
| `load("`*fichier*`")` | idem sans écho ou chargement d'une biblio (ex. `load("eigen")`). |
| `display2d`[`true`] | `false` pour affichage sous forme de chaîne réutilisable. |
| `ibase`[10] et `obase`[10] | base utilisée pour les entrées et sorties des nombres. |

## Généralités

| | |
|---|---|
| *expr*`;` ou *expr*`$` | caractère de fin pour évaluer avec (`;`) ou sans (`$`) sortie écran. |
| `%th(`*i*`)` | le $i^e$ dernier calcul; `%` = `%th(1)`. |
| `'`*expr* | expression non évaluée; par ex. `'`$f(x)$ ou `'`$(f(x))$. |
| `''`*expr* | expression réévaluée; par ex. `''`$f(x)$, `''`$(f(x))$, `''C`$i$ ou `''`$x$ (2 évaluations). |
| *ident*`:`*expr* ou *expr*`::`*expr* | affectation; *ident* est sensible à la casse; valeur: *expr*. |
| *ident*`(`*args*`) := `*expr* | définition de fonction; ou *ident*`[`*args*`] := `*expr*: déf. récursive de tableaux. |
| `kill(`*arg1*`,...)` | *argi* sont des noms de variables, fonctions ou tableaux ou `all`. |

`alias(`*newname1*`,`*oldname1*`,...)` nouveaux noms pour des fonctions, variables, tableaux, etc.

## Expressions

| | |
|---|---|
| *contrôles* `do` *expr* | boucle; valeur renvoyée: `done` ou `return(`*valeur*`)`; les contrôles sont: |
| `for` *var* (`:`—`from`—`in`) *expr* | val. initiale (1 par déf.) si `:`—`from`; si `in`, *var* ∈ {opérandes de *expr*}. |
| (`step`—`next`) *expr* | contrôle de la variable; `step` $p$ ⇔ `next` *var* + $p$. |
| `thru` *expr* | valeur à ne pas dépasser pour la variable de contrôle. |
| (`while`—`unless`) *expr* | `while` *expr* ⇔ `unless not` *expr*. |
| `if` *expr* `then` *expr* [`else` *expr*] | opérateurs: `= # < > <= >= and or not`. |
| `block([`*var1*[`:`*e1*]`,...],`*expr1*`,...)` | les *vari* sont locales; valeur: dernière *expri* ou `return(`*val*`)`; |
| `local(`*var1*`,...)` | fonctions locales ou tableaux locaux; |
| `...,`*tag*`,...,go(`*tag*`),...` | par ex. `block([x],x:1,loop,x:x+1,...,go(loop),...)`. |
| `ev(`*expr*`,`*arg*$_1$`,...)` | évaluation de *expr*; les *arg*$_i$ sont: |
| des liaisons locales *ident*`:`*expr* (ou `=`) effectuées en parallèle; | |
| des listes d'équations `[`*ident*$_1$`=`*expr*$_1$`,...]` traitées comme des liaisons locales; | |
| des définitions locales de fonctions $F(args) := \ldots$; | |
| des atomes à true pdt l'évaluation: `simp`, `numer`, `float`, `pred`, etc. (`evflag`); | |
| des fonctions à appliquer, dans l'ordre, à *expr*: `factor`, `bfloat`, etc. (`evfun`); | |
| des noms de fcts à évaluer ds *expr*: `diff`, etc.; `nouns` pour toutes. | |
| `expand`, `expand(`*exposant*$>0$`,`*exposant*$<0$`)`. | |
| *expr*`,`*arg*$_1$`,...` | idem mais pas comme sous expression. |
| `(`*expr*$_1$`,...,`*expr*$_n$`)` | séquence; valeur: *expr*$_n$. |
| `catch(`*expr*$_1$`,...,`*expr*$_n$`)` | renvoie $e$ si `throw(`$e$`)` est évalué. |

## Types

prédicats: `atom`, `listp`, `matrixp`, `numberp`, `symbolp`, `unknown`, `bfloatp`, `constantp`, `evenp`, `floatnump`, `integerp`, `nonscalarp`, `oddp`, `primep`, `ratnump`, `ratp`, `scalarp`, `subvarp`, `taylorp`.

fonctions de conversion: `bfloat`, `entier`, `float`.

## Simplification

| | |
|---|---|
| `radcan(`*expr*`)` | simplifie *expr* qui peut contenir des logs, des exp, des radicaux. |
| `expand(`*expr*[`,`*exp*$>0$[`maxposex`] `,`*exp*$<0$[`maxnegex`]]); indépendamment de `expand`: `expop`[0] et `expon`[0]. | |
| `partfrac(`*expr*`,`*var*`)` | décomposition en éléments simples. |
| `combine(`*expr*`)` | regroupe les termes ayant le même dénominateur. |
| `ratexpand(`*expr*`)` | ⇒ $\sum_i p_i/d$, $p_i$ monomes, $d$ pnome développé. |

| | |
|---|---|
| `rat[simp](`*expr*`[,`$v_1,\ldots$`])` | $\Rightarrow p/q$, $p$ et $q$ pnomes développés en f. de `ratvars(b,a,x)` ou `ratvars(x)`. |
| `xthru(`*expr*`)` | $\Rightarrow p/q$, $p$ et $q$ pnomes non développés. |
| `tellrat(`$p(a)$`)`, $p$ pnome | si `algebraic` = `true`, déclare $a$ racine de $p$; alors `rat(`*expr*`)`. |
| `trigexpand(`*expr*`)` | dév. complet: `trigexpand(`*expr*`),trigexpand=true,expand;` |
| `trigreduce(`*expr*`)` | linéarise (utiliser aussi `declare(j,integer,e,even,o,odd)`). |
| `trigsimp(`*expr*`)` | simplifie par $\cos^2 x + \sin^2 x = 1$ et $\cosh^2 x - \ldots$ |
| `logexpand[true]` | `true`: $\log a^b \Rightarrow b \log a$, `all`: $\log(ab) \Rightarrow \ldots$, `super`: $\log(a/b) \Rightarrow \ldots$. |
| `logcontract(`*expr*`)` | contraire de `logexpand`. |
| `demoivre[false]` | $e^{a+ib} \Rightarrow e^a(\cos b + i \sin b)$; ou `demoivre(`*expr*`)`. |
| `exponentialize[false]` | fcts circ. et hyperb. $\Rightarrow$ forme exp; ou `exponentialize(`*expr*`)`. |
| `load("atrig1")` | $\arccos(1/\sqrt{2}) \Rightarrow \ldots$. `load("ntrig")` pour $\cos(\pi/10) \Rightarrow \ldots$. |
| `logarc[false]` | fcts hyperbol. réciproques $\Rightarrow$ forme log; ou `logarc(`*expr*`)`. |
| `factor(`*expr*`[,`*pnome-min*`])` | ou `factor(`*expr*`),dontfactor:[`*vars*`];`; dans **C**: `gfactor`. |
| `polarform(`$z$`)` et `rectform(`$z$`)`. | |

**Opérandes, substitution**

| | |
|---|---|
| `reveal(`*expr*`,`*profondeur*`)` | structure de *expr*. |
| `dispterms(`*expr*`)` | affiche l'opérateur puis les opérandes de *expr* sur des lignes séparées. |
| `[d]part(`*expr*`,`$n_1,\ldots,n_k$`\|[`$n_k^1$`,.])` | sous expression(s) (stockée ds la var. `piece`); `dpart` pour voir. |
| `first(`*expr*`)`, `last(`*expr*`)` | `part(`*expr*`,1)`, `part(`*expr*`,length(`*expr*`))`. |
| `rest(`*expr*`,`$n$`)` | supprime les $n$ $1^{\text{ers}}$ ou les $-n$ derniers élts; $n = 1$ par déf. |
| `pickapart(`*expr*`,`$n$`)` | labelise (`E`$i$) les sous expressions de profondeur $n$. |
| `lhs(`*eqn*`)` et `rhs(`*eqn*`)` | les 2 membres d'une équation. |
| `realpart(`$z$`)` et `imagpart(`$z$`)` | complexes (marche sur les fcts transcendantes). |
| `num(`*fract*`)` et `denom(`*fract*`)` | `ratnum[er]` et `ratdenom` pour forme CRE. |
| `[rat]coeff(`*expr*`,`$v$`[,`$n$`])` | coeff. de $v^n$ dans *expr*; `ratcoeff` développe et simplifie *expr*. |
| `hipow(`*expr*`,`$v$`)`, `lopow(`*expr*`,`$v$`)` | et `powers(`*expr*`,`$v$`)` degré, valuation et liste des exposants (développer). |
| `substpart(`*new*`,`*expr*`,`$n_1,\ldots$`)` | substitution de sous expression; ex. `liste(e):= substpart("[",e,0);`. |
| `[rat]subst(`*new*`,`*old*`,`*expr*`)` | ou `subst([`$old_1$`=`$new_1,\ldots$`],`*expr*`)` en série; `ratsubst` + efficace. |

**Listes**

| | |
|---|---|
| `[`$élt_1,élt_2,\ldots$`] length(`*liste*`)`. | |
| `makelist(`*ei*`,`*i*`,`*min*`,`*max*`)` | ou `makelist(`*ei*`,`*i*`,`*liste*`)` $\Leftrightarrow$ `map(lambda([`*i*`],`*ei*`),`*liste*`)`. |
| `map(`*fct*`,`*expr*`)` | remplace ch. opérande de *expr* par son image par *fct*. |
| `map(`*fct*`,`*expr1*`,...,`*exprn*`)` | `map(f,y+x,z+t)` $\Rightarrow$ `f(y, z) + f(x, t)`. |
| `maplist(`*fct*`,`*expr1*`,...,`*exprn*`)` | `maplist(f,y+x,t*z)` $\Rightarrow$ `[f(y,t),f(x,z)]`. |
| `[end]cons(`*expr*`,`*liste*`)` | ajoute *expr* en début [ou en fin] de *liste* (marche sur tte expression). |
| `append(`$liste_1$`,`$liste_2,\ldots$`)` | concaténation (marche sur tte expression). |

**Maths**

| | |
|---|---|
| `%e`, `%i`, `%pi`, `%gamma`, `%phi` | constantes usuelles; `inf` $= \infty$, `infinity` $= \infty \in \overline{\mathbf{C}}$. |
| `abs(`$x$`)`, `cabs(`$z$`)`, `carg(`$z$`)`, `max(`$x_1,\ldots$`)`, `min(`$x_1,\ldots$`)`, `signum(`$x$`)`, `sqrt(`$x$`)`, `exp(`$x$`)`, `log(`$x$`)`, `plog(`$z$`)`. | |
| `[a]cos[h](`$x$`)`, `[a]sin[h](`$x$`)`, `[a]tan[h](`$x$`)`, `[a]cot[h](`$x$`)`, `atan2(`$y$`,`$x$`)`, `gamma(`$x$`)`, $n$`!`. | |
| `bfloat(`*expr*`)` | approché à `fpprec` près; `float(bfloat(`*expr*`))` pour une courbe par ex. |
| `solve([`$e_1$`,..],[`$x_1$`,..])` | `[[`$x_1$`=`$v_1^1,\ldots$`],[`$x_1$ `=` $v_1^2,\ldots$`]]` ou `solve(`$e$`,`$x$`)` $\Rightarrow$ `[`$x$ `=` $v^1$`,`$x$ `=` $v^2,\ldots$`]`. |
| `[t]limit(`*expr*`,`$x$`,`$a$`[,`*dir*`])` | $\infty$ = `inf`, $-\infty$ = `minf`; *dir* = `plus` \| `minus`; `tlimit` pour util. `taylor`. |
| `taylor(`$f(x)$`,`$x$`,`$a$`,`$n$`)` | DL; `taytorat(`*dl*`)` pour convertir en CRE. |
| `deftaylor(`$f(x)$`,`*expr*`)` | *expr* est un pnome ou `sum(...)`. |
| `powerseries(`$f(x)$`,`$x$`,`$a$`)` | Dév. en SE de $f$ en $a$; `verbose:true` pour explications |
| `depends(`$fcts_1$`,`$vars_1,\ldots$`)` | les args sont des (listes de) noms qui définissent les dépendances pour `diff`. |
| `diff(`*expr*`,`$v_1$`,`$n_1,\ldots$`)` | ou `diff(`*expr*`,`$v$`)` ou `diff(`*expr*`)` (dérivées en f. de `del`). |
| `integrate(`$e$`,`$x$`[,`$a$`,`$b$`])` | pour une v.a. utiliser `romberg` ou, + efficace: `describe(romberg)`. |
| `changevar(`$e$`,`$f(x,u)$`,`$u$`,`$x$`)` | $f(x,u) = 0$ ds ttes les intégrales de fct de $x$ ds $e$. |
| `[nu]sum(`*expr*`,`*i*`,`*min*`,`*max*`)` | `nusum` pour Gosper; `simpsum:true` p. simplifier (bug!). Idem `product`. |

`depends(y,x)` appeler avant d'utiliser `ode2`.
`ode2(`*eqn*`,y,x)` *eqn* est d'ordre 1 ou 2; utiliser `x`, `y`, `diff(y,x)` et `diff(y,x,2)`.
`ic1(`*soln*`,x=`$x_0$`,y=`$y_0$`)` cdns initiales; `ic2(`*soln*`,x=`$x_0$`,y=`$y_0$`,'diff(y,x)=`$y_1$`)` si ordre 2.
`bc2(`*soln*`,x=`$x_1$`,y=`$y_1$`,x=`$x_2$`,y=`$y_2$`)` conditions aux limites pour une équation d'ordre 2.
`atvalue(`*expr*`,x=`$x_0$`,`$e_0$`)` *expr* $= e_0$ en $x_0$; *expr* $= f$`(x)` ou `'diff(`$f$`(x),x[,2])`.
`desolve([`$e_1$`,...],[`$f$`(x),`$g$`(x),...])` $e_i$ en fct de $f$`(x)` et `'diff(`$f$`(x),x[,2])`.

### Polynomes

`divide(`$p$`,`$q$`,`$y_1$`,...,`$y_n$`,`$x$`)` = `[quotient(...),remainder(...)]`.
`resultant(`$p$`,`$q$`,`$x$`)` = `determinant(bezout(`$p$`,`$q$`,`$x$`))`.
`gcd(`$p$`,`$q$`,`$x$`)` `algebraic`[`false`] doit être `true` si des coeffs sont algébriques ĉ $\sqrt{2}$.
`lcm(`$p_1$`,...,`$p_n$`)` ppcm; accés par `load("functs");`.
`content(`$p$`,`$y_1$`,...,`$y_n$`,`$x$`)` = `[`$c$ (*contenu de* $p$)`,`$p/c$`]`.
`allroots(`$p$`)` valeurs approchées des racines réelles et complexes.

### Algèbre linéaire

`array(`[*liste de*]*nom*`,`$d_1$`,...,`$d_n$`)` déclare un [des] tableau; $n \leqslant 5$, indices $\in [0, d_i]$.
*nom*`[`$i_1$`,...,`$i_n$`]` élt d'1 tableau déclaré ou non (auq. cas 1 seul indice mais qcq).
`matrix(`$\ell_1$`,...,`$\ell_p$`)` déf. d'une matrice $p \times q$: $\ell_i =$ `[`$a_{i,1}$`,...,`$a_{i,q}$`]`.
`genmatrix(`$t$`,`$p$`,`$q$`)` déf. d'une matrice $p \times q$: $t =$ `lambda([`$i$`,`$j$`],`*expr*`)` ou $t$`[`$i$`,`$j$`]` `:=` *expr*.
`ident(`$p$`)` $I_p$; $\lambda I_p =$ `diagmatrix(`$p$`,`$\lambda$`)`.
`zeromatrix(`$p$`,`$q$`)` $0_{\mathcal{M}_{p,q}(\mathbf{K})}$; $\lambda E_{i,j} =$ `ematrix(`$p$`,`$q$`,`$\lambda$`,`$i$`,`$j$`)`.
`row(`$A$`,`$i$`)` et `col(`$A$`,`$j$`)` renvoie une matrice ligne ou colonne; $A[i, j]$ pour 1 élt.
`submatrix(`$i_1$`,...,`$A$`,`$j_1$`,...)` suppression des lignes $i_1, \ldots$ et des colonnes $j_1, \ldots$.
`addrow(`$A_1$`,..) addcol(`$B_1$`,..)` concaténation: les $A_i$ ($B_i$) doivent avoir m̂ nombre de colonnes (lignes).
`ratmx`[`false`] si `true`, les coefficients des matrices auront la forme CRE.
$A$ `.` $B$, $\lambda$ `*` $A$ et $A$`^^`$n$ $A$`*`$B$, $\lambda$`^`$A$ et $A$`^`$\lambda$ opèrent élt. par élt; `transpose(`$A$`)`.
`invert(`$A$`)` et `adjoint(`$A$`)` $A^{-1}$ et ${}^{\mathrm{t}}\mathrm{cof}A$; ex: `expand(invert(`$A$`)),detout; xthru(%);`.
`rank(`$A$`)` et `determinant(`$A$`)` `determinant(`$A$`),ratmx,sparse` pour une matrice creuse.
`charpoly(`$A$`,`$\lambda$`)` ou `ncharpoly` si `load("nchrpl.mac")` (qui donne aussi `mattrace(`$A$`)`).

`load("eigen");` pour les fonctions suivantes:
`eigenvalues(`$A$`)` `[[`$\lambda_1$`,...,`$\lambda_p$`],[`$m_1$`,...,`$m_p$`]]`, $\lambda_i$ vp de multiplicité $m_i$.
[`unit`]`eigenvectors(`$A$`)` `[eigenvalues(`$A$`),[`$v_1^1$`,...],...]`, affecte le booléen `nondiagonalizable`.
`similaritytransform(`$A$`)` = `uniteigenvectors(`$A$`)` et si diagonalisable, $LAR = D$ et $LR = I$ où $L =$ `leftmatrix` et $R =$ `rightmatrix` ($L = R^*$ si `hermitianmatrix`).
`conjugate(`$A$`)` `%i` doit être explicite.
`innerproduct(`$X$`,`$Y$`)` listes ou matrices ligne ou colonne; aussi `unitvector(`$X$`)`.